**PROPRIETARY NOTICE**

All rights reserved by Janz Automationssysteme AG.
No parts of this technical manual may be modified, copied or reproduced in any form or by any means for commercial use without the prior written permission of Janz Computer, Germany.
All instructions, information and specification contained in this manual are for reference only and remain subject to change without announcement.

# emPC-X133

fanless computer system with CAN and Ethernet interface

(Hardware Manual)

Version 3

refers to product revision no.
V 2.x

| | |
|---|---|
| Title: | emPC-X133 Manual |
| File: | y:\empc\empc_x133_megabox\rev1\manual\canmbx_man3.doc |
| Pattern: | \\ntserv1\entwicklung (ew)\formblätter\entwicklung (ew)\f_manual.dot |
| Created: | hr, 06.01.2005 |
| Last Update: | hr, 06.01.2005 |

© Janz Automationsysteme AG 2001

Im Dörener Feld 8
PO Box 1906
D-33 049 Paderborn, Germany

| | |
|---|---|
| Tel.: | +49-5251-1550-0 |
| FAX: | +49-5251-1550-90 |
| email: | support@janz.de |
| Internet: | www.janz.com |

# Contents

**1 Introduction 5**

1.1 Hardware ....5
1.2 Software ....6
1.3 Functional Overview ....6

**2 Installation 7**

2.1 Front Panel , connectors and LEDs ....8
2.1.1 RS232 connector “serial” ....9
2.1.2 CAN-bus connector ....10
2.1.3 Ethernet Interface ....10
2.1.4 emPX-X133 system status LEDs and Reset ....10
2.1.5 External IO-connector ....11
2.1.6 Power supply connector ....11
2.1.7 USB type A receptable ....11

**3 Internal System Hardware 12**

3.1 CAN interface ....12
3.2 I/O interface ....13
3.3 User programmable Front Panel LEDs ....14
3.4 CompactFlash Socket ....15
3.5 USB Host controller ....15
3.6 $I^2C$ EEPROM ....15
3.7 $I^2C$ temperature sensor LM75 ....18
3.8 Interrupt mapping and PCI device numbers ....18
3.9 Memory Map ....18

**4 SrecLOAD Firmware 20**

4.1 Using SrecLOAD ....20
4.1.1 Invoking SrecLOAD ....20
4.1.2 Downloading Programs/Files ....20
4.1.3 Command Mode ....21
**4.1.3.1 configuring the boot up process ....22**
4.1.4 Autoboot ....23
4.2 Details about the Download Function ....23

**5 Appendices 26**

5.1 Technical Data ....26
5.2 References ....27
5.3 MOTOROLA S-Record Format ....28
5.3.1 S-Record Content ....28
5.3.2 S-Record Types ....28
5.3.3 S-Records Creation ....29
5.3.4 Example ....29
5.4 Product History ....31
5.5 Manual History ....31

 **© Janz Automationsysteme AG**

# List of Figures

figure 1: emPC-X133/1 ....5
figure 2: emPC-X133 “basic” block diagram....6
figure 3: simple null modem cable....7
figure 4: front panel with IO-port, emPC-X133/1 ....8
figure 5: front panel with USB, emPC-X133/1U ....8
figure 6: emPC-X133/2 (with additional CAN and IO-Port) ....8
figure 7: emPC-X133/2U (with two additional CAN ports)....9
figure 8: emPC-X133/3 (with two additional CAN ports) ....9
figure 9 : optical decoupled IO-port circuit....13
figure 10: FLASH layout ....24

# List of Tables

table 1: emPC-X133 versions....iv
table 2: additional features ....iv
table 3: serial port....9
table 4: CAN connector ....10
table 5: Ethernet connector ....10
table 6: Ethernet controller status LEDs....10
table 7: system status LEDs....11
table 8: external IO-connector ....11
table 9: power supply connector....11
table 10: USB connector....11
table 11: EEPROM timing....16
table 12: EEPROM address map ....18
table 13: interrupts and PCI devices ....18
table 14: system memory map ....19
table 15: SrecLoad boot hook details....25

# Conventions

If numbers are specified in this manual, they will be either decimal or hexadecimal. We use C-notation to identify hexadecimal numbers (the 0x prefix).

Register descriptions are done in the following style:

*Word address* ↓ **0x080**, *Access width* ↓ **(byte,**, *Access type* ↓ **rw)**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Register Name* → | serial EEPROM | | | | | | | |
| Bit Numbers → | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| *Field Numbers* → | reserved | | | | | | SCL | SDA |
| | Reset: | | | | | | | |
| *State after Reset* → | - | | | | | | 0 | 0 |

The *access type* specifies the possible operations on this register. The code **ro** (read-only) says that this register is only defined for read operations, while the code **wo** (write-only) indicates that this register can only be written. Undefined operations must be performed on a register.
The code **rw** (read/write) specifies that this register might be read and written. Depending on the detailed description for a register, this does not necessarily mean that you can do read-modify-write operations on this register.

The *state-after-reset* specifies the value that is read after reset, or the internal value after reset if the register is of type read-only. If several reset conditions are possible, the exact source is specified.

Some parts of the text are really important. These are visually marked with the following signs:

Indicates information that, when not fully understood or followed, might cause permanent damage to the system. You should not start using the product before you have read this information.

Indicates information that we think you should have read to save your time by avoiding problems. Important suggestions that should be followed will also be marked with this sign.

# Acronyms and Abbreviations

| | |
|---|---|
| ESD | Electrostatic discharge. |
| FLASH | Electrically erasable PROM. Capable of in-circuit re-programming with the capability of erasing considerably large blocks (in contrast to EEPROM). |
| NVRAM | Non volatile RAM. Storage is ensured by a battery. |
| USB | Universal Serial BUS |
| PCI | Peripheral Component Interconnect. |
| EEPROM | Electrically erasable PROM. Capable of individually in-circuit re-programming of each cell (in contrast to FLASH) |

# Product Options

The emPC-X133 V2.x is available in five different versions:

| order-no. | Type |
|---|---|
| CB-MBX-01001 | emPC-X133/1, smallest form factor, IO-Port |
| CB-MBX-01002 | emPC-X133/1U smallest form factor, no IO-port, but USB 1.1 host support |
| CB-MBX-02001 | emPC-X133/2 with additional CAN and IO-Port |
| CB-MBX-02002 | emPC-X133/2U with additional CAN and USB 1.1 host |
| CB-MBX-02003 | emPC-X133/3 with 3 CAN interfaces |

table 1: emPC-X133 versions

Additionally the emPC-X133 has the following optional features. Check the price-list for details. All of these options are build-to-order, and cannot be updated by the end-user.

| Feature | Added Feature |
|---|---|
| OP-MBX-MM064 | Memory option: 64 MB internal SDRAM |
| OP-MBX-MM128 | Memory option: 128 MB internal SDRAM |
| OP-MBX-CF064 | Data storage option: 64 MB Compact flash card |
| OP-MBX-CF128 | Data storage option: 128 MB Compact flash card |
| OP-MBX-CF256 | Data storage option: 256 MB Compact flash card |
| OP-MBX-CF512 | Data storage option: 512 MB Compact Flash card |

table 2: additional features

Note: The emPC-X133 was formerly known as CAN-megaBox.

# 1 Introduction

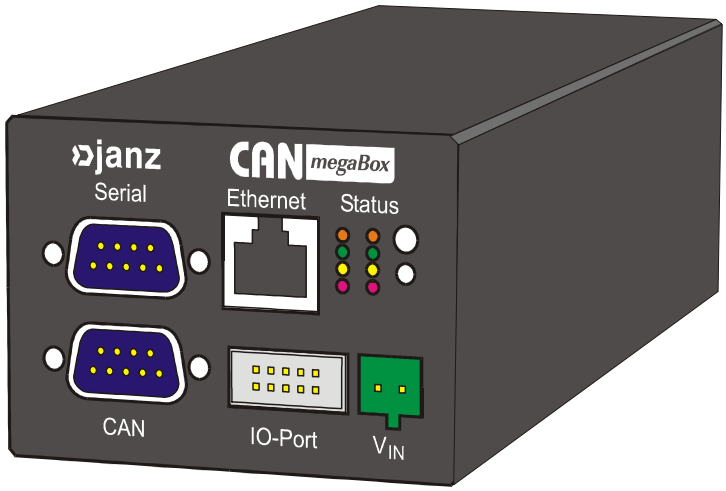



figure 1: emPC-X133/1

The emPC-X133 combines a high performance computer with small dimensions. Because of the small outline of the housing and the mounting clips onto the back side it can be installed in nearly every location with a rail mount system. The emPC-X133 only needs an external power supply of 10 .. 28 V (AC or DC).
Due to it's build in x86 compatible CPU, a wide wide range of OSes like Linux, OS/9 or Windows CE can be run.The user can create his own application for use with the emPC-X133 on his desktop PC and, when finished, the program can be transferred to the emPC-X133 via Ethernet or serial port.

Features

## 1.1 Hardware

- 133MHz x86 compatible AMD Élan SC520 CPU
- a maximum of 128 MB SDRAM (32bit memory bus) with 66MHz bus clock
- 2MB on board boot flash (e.g. for containing the Linux boot kernel)
- internal compact flash socket type I/II for "true-IDE mode" compatible compact flash cards
- 10 / 100 MBit AM79C973 PCI Ethernet controller
- up to three CAN interfaces with individual SJA1000 CAN controller
- RS-232 serial port (16550 compatible)
- up to two general purpose IO-ports, each with 4 optical decoupled Inputs and 4 optical decoupled outputs for 5V or 24V level (ordering option, default is 24V)
- optional USB 1.1 interface using OPTi 82C861 PCI USB controller (compatible to OHCI standard) 4 frontpanel LEDs indicate the status of the network controller
- 1 status LEDs for system
- 3 user programmable LED
- integrated real time clock with 114 bytes non volatile SRAM and battery backup
- 4kBit serial EEPROM for non volatile user data
- internal power regulation, only a 10 .. 28V (AC or DC) power source is needed
- internal LM75 temperature sensor with programmable thermal watchdog function

## 1.2 Software

- Supported by Linux (VxWorks, Win-CE driver in preparation)

## 1.3 Functional Overview

The emPC-X133 uses an AMD Elan SC520 microcontroller. This chip combines a 32-bit AM 5x86 CPU with a set of integrated peripherals suitable for PC/AT compatible embedded applications.

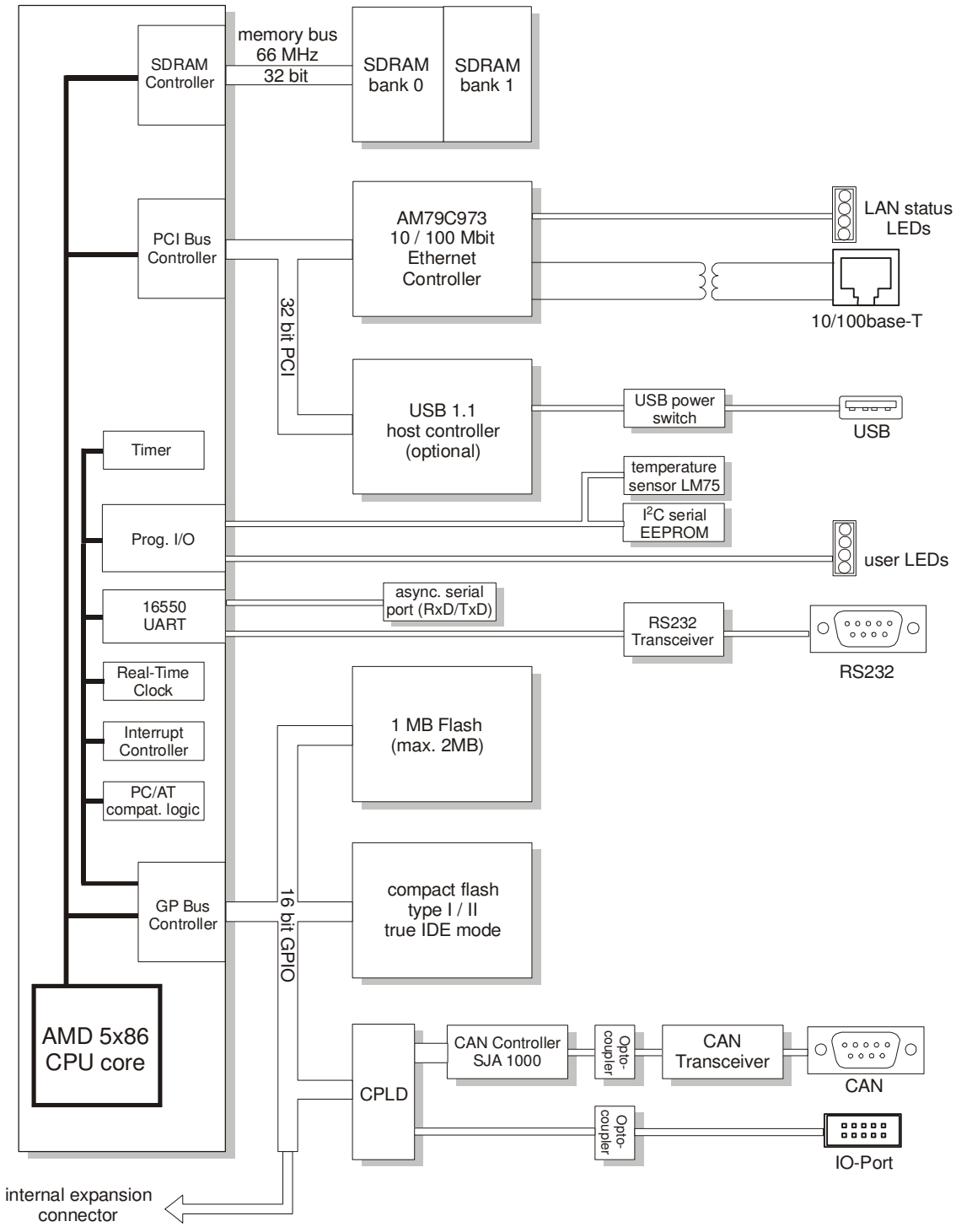



figure 2: emPC-X133 “basic” block diagram

# 2 Installation

When opening the shipping package of the emPC-X133, you should immediately check the contents of the package. In the package you will find information about the scope of delivery, as this depends on the options that you have ordered.

If not otherwise noted, the emPC-X133 is delivered in a properly configured state. That is, memory and flash disk are installed. The boot flash will at least host the SrecLOAD software that is needed to install programs in the flash. If your order included support for an operating system, this will be installed in a basic manner.

The package should also contain a terminal plug for the power connector. The emPC-X133 needs a power source of 10..28 V AC or DC for operation. You can easily connect the wires of your power supply to the terminal plug. When powering up the emPC-X133, the green Power-LED should light (see chapter 2.1).

The housing is equipped with mounting clips. This allows the emPC-X133 to be installed on every mounting rail which is compatible to EN 50022.

Since the emPC-X133 has no VGA adapter, boot messages are only transmitted via the serial port. You need a host PC with a terminal emulator program running (e.g. freeware version of TeraTerm, © by T. Teranishi) and a null modem connection between both systems (figure 3).
If you have ordered the emPC-X133 with a preinstalled Linux, our RLL application (ROM Linux Loader) is already programmed into the boot flash. You should make the following settings in your terminal emulator:

- speed = 9600 baud
- 8 databits, no parity bit, 1 stop bit
- no flow control

To use our SrecLoad firmware, the following settings must be made for a fast download:

- speed = 115200 baud
- 8 databits, no parity bit, 1 stop bit
- no flow control

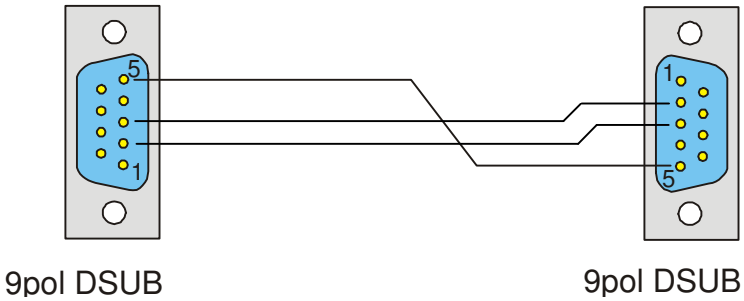



figure 3: simple null modem cable

## 2.1 Front Panel , connectors and LEDs

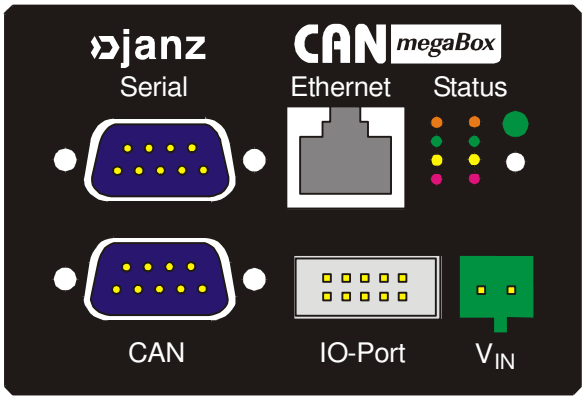



figure 4: front panel with IO-port, emPC-X133/1

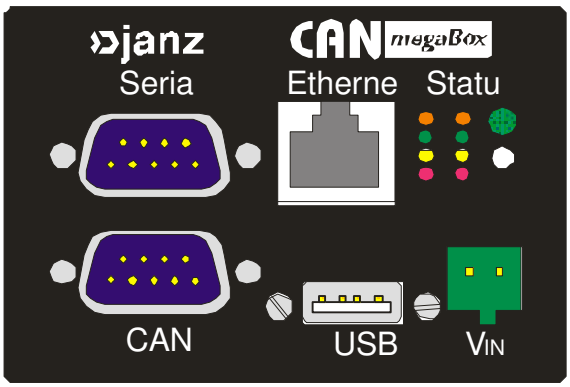



figure 5: front panel with USB, emPC-X133/1U

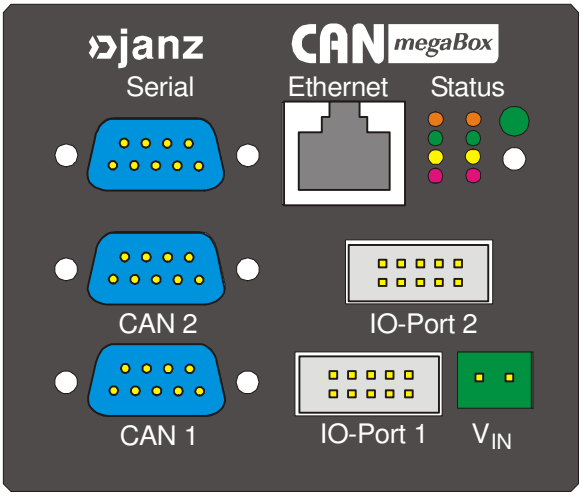



figure 6: emPC-X133/2 (with additional CAN and IO-Port)

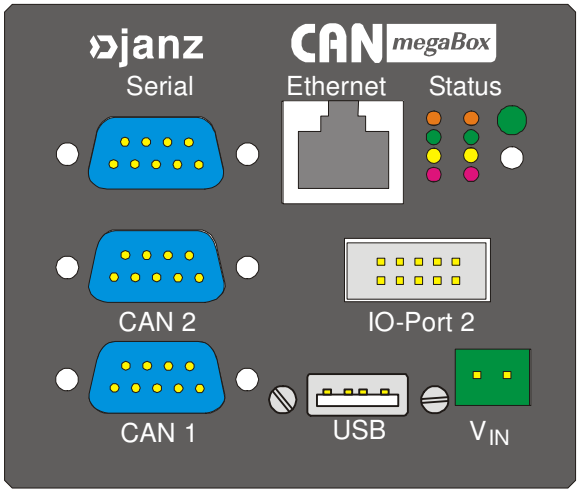



figure 7: emPC-X133/2U (with two additional CAN ports)

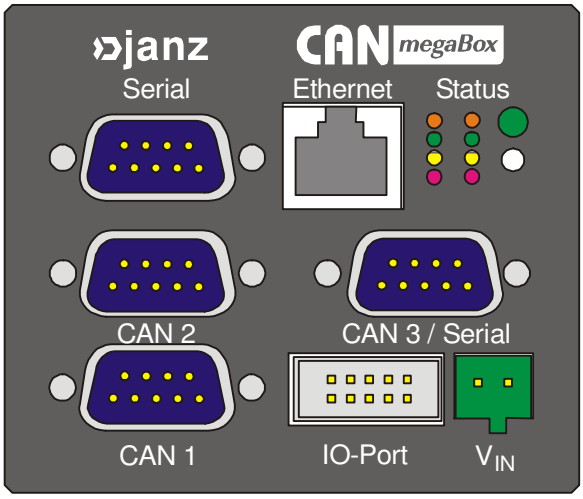
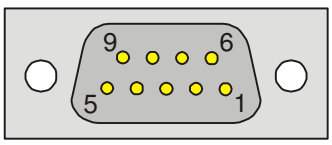



figure 8: emPC-X133/3 (with two additional CAN ports)

The figures above show the front panel views of the five different versions of the emPC-X133.

The serial RS232 and the CAN-Bus both have the same type of connectors! When installing the emPC-X133 these connectors can easily be mixed up! Please refer to the above figures if you are unsure.

### 2.1.1 RS232 connector "serial"

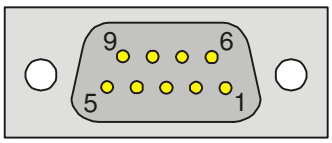



| | | | |
|---|---|---|---|
| 1 | DCD | 6 | DSR |
| 2 | RxD | 7 | RTS |
| 3 | TxD | 8 | CTS |
| 4 | DTR | 9 | n.c. |
| 5 | GND | | |

table 3: serial port

This is the standard pinout for the 9 pol DSUB male RS 232 connector. If you want to connect the emPC-X133 to another PC you need a null modem cable with crossed RxD, TxD lines.

### 2.1.2 CAN-bus connector

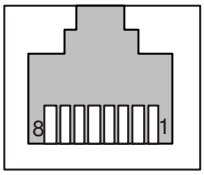



| | | | |
|---|---|---|---|
| 1 | n.c. | 6 | ext. GND |
| 2 | CAN_L | 7 | CAN_H |
| 3 | ext. GND | 8 | n.c. |
| 4 | n.c. | 9 | n.c. |
| 5 | n.c. | | |

table 4: CAN connector

### 2.1.3 Ethernet Interface

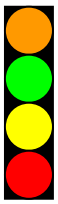
The emPC-X133 provides a 10/100Mbit Ethernet interface, so that it can be integrated into local area networks. The Ethernet controller (AMDs AM79C973) is hosted on the PCIbus. It works as a PCIbus master, that directly writes incoming packages to the main memory. Data for outgoing packages is fetched from main memory without any CPU interaction.
The Ethernet physics is 10/100BaseT, available through the shielded modular jack at the front panel. Category-5 twisted pair cable can be used to connect to this port:




| | | | |
|---|---|---|---|
| 1 | TD+ | 5 | n.c. |
| 2 | TD- | 6 | RD- |
| 3 | RD+ | 7 | n.c. |
| 4 | n.c. | 8 | n.c. |

table 5: Ethernet connector

The four LEDs next to the Ethernet connector indicate the LAN-Controller status.



Orange
Green
Yellow
Red

| | | |
|---|---|---|
| orange | LAN speed | indicates a connection to a 100MBit network when lit. |
| green | Link status | When lit, this LEDs indicates that the emPC-X133 is connected to a LAN |
| yellow | LAN activity | flashes if packets are transmitted and received |
| red | collision detect | indicates a packet collision on the LAN is detected |

table 6: Ethernet controller status LEDs

### 2.1.4 emPX-X133 system status LEDs and Reset

Orange
Green
Yellow
Red

System Reset Button

System Power

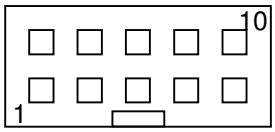
| | |
|---|---|
| Orange | this LED shows the activity of the internal Flash disk (not always supported, depends on the manufacturer / type of the installed flashdisk) |
| System Power | the System Power LED indicates that all internal power supply stages are operational |

| | |
|---|---|
| System Reset | When pressing this button, the system performs a hard reset |
| green, yellow, red | these LEDs are programmable by the user. Only during the power up sequence these LEDs have special meanings. Refer to chapter 4 for more information. |

table 7: system status LEDs

### 2.1.5 External IO-connector

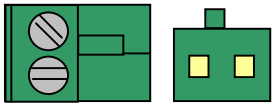
This simple user IO-port is designed for applications that use a 24V level.




| | | | |
|---|---|---|---|
| 1 | $V_{OUT}$ 0 | 6 | Ext. GND |
| 2 | $V_{OUT}$ 1 | 7 | $V_{IN}$ 0 |
| 3 | $V_{OUT}$ 2 | 8 | $V_{IN}$ 1 |
| 4 | $V_{OUT}$ 3 | 9 | $V_{IN}$ 2 |
| 5 | Ext. power supply $V_{EXT}$ | 10 | $V_{IN}$ 3 |

table 8: external IO-connector

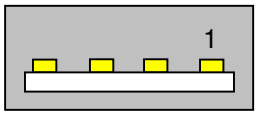
### 2.1.6 Power supply connector



Power supply connector with terminal plug.
Apply 12 .. 24 V DC or AC for operation

table 9: power supply connector

### 2.1.7 USB type A receptable




| | |
|---|---|
| 1 | $V_{BUS}$ |
| 2 | Data - |
| 3 | Data + |
| 4 | GND |

table 10: USB connector

# 3 Internal System Hardware

The emPC-X133 is a complete stand alone computer system. Due to its build in PC hardware compatible components, almost all Intel™ x86 compatible software can be used (remember that the "Basic" version of the emPC-X133 does not support VGA graphics and multiple serial ports).
The main processor is an AMD Elan SC520™ microcontroller with integrated peripherals.
Here are the hardware features in detail:

- 5x86 CPU core running at a clock frequency of 100 or 133 MHz
- integrated floating point unit (compliant with IEEE 754)
- 16 kB unified L1 cache
- 33 MHz, 32 bit PCI bus interface (PCI Rev. 2.2 compliant)
- 66 MHz, 32 bit SDRAM interface (max. 128 MB is supported by the emPC-X133)
- Flash controller, up to 2MB 16bit flash is supported
- PC/AT compatible programmable interrupt controller
- PC/AT compatible programmable interval timer that consists of three 16 bit timers
- real-time clock with battery backup and 114 bytes battery-backed RAM
- 2 integrated 16550-compatible UARTs (only one UART is externally usable with the emPC-X133)
- compact flash (IDE mode) interface (realized via the SC520's general purpose bus) with an internal type I socket connector.
- CAN interface with the Philips SJA1000 CAN controller. The SJA1000's registers are mapped into memory space at address `0xd0000` (`0xd1000` and `0xd2000` for additional CAN ports)
- user IO-connector with 4 opto-isolated input lines and 4 opto-isolated output lines suited for 5V user applications. The IO-registers are located within the IO-space at address `0x300`
- three user programmable front panel LEDs
- 10 / 100 MBit AM79C973 Ethernet controller with internal MAC and PHY (connected to the PCI bus using PCI interrupt D (IRQ 10))
- USB 1.1 host controller OPTi 82C861 Fire Link™ (connected to the PCI bus using interrupt A (IRQ 11))
- internal $I^2C$ bus connected to an EEPROM (24LC04) and to the temperature sensor LM75 (with thermal watchdog function)

## 3.1 CAN interface

The integrated CAN interface of the emPC-X133 uses the SJA 1000 stand-alone CAN controller manufactured by Philips Semiconductors. For more information about this device please refer to the SJA 1000 Data Sheet [3]. There you can find a complete register description you will need, if you want to program own applications.
All the controller's registers are mapped into a 4 kB memory area, starting at `0xd0000` for the first CAN interface. A second CAN interface (optional) can be accessed at `0xd1000` and the memory area starting at `0xd2000` is reserved for a third CAN interface port. In the following the number of the CAN port is substituted by X in the address numbers.
Since not all address lines are decoded, the registers are repeated within this 4 KB memory region.
It is recommended to use the memory addresses starting at `0xdX000`. If you use the BasiCAN mode of the SJA1000 controller the valid address area is `0xdX000 .. 0xdX01f`. The PeliCAN mode uses the address region from `0xdX000 .. 0xdX07f`.

In some cases it could be important to terminate the CAN line. The emPC-X133 has a built in software controlled CAN termination. When activated, a 120Ohms resistor is switched between both CAN lines.

CAN bus termination **0xdX100 (byte, r/w)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| reserved | | | | | | | TERM |
| RESET: | | | | | | | |
| | | | - | | | | 0 |

TERM = 0 bus termination disabled
TERM = 1 bus termination enabled

The CAN Controller is able to generate interrupts. The interrupt signal is mapped to the IRQ5 line.

## 3.2 I/O interface

The emPC-X133 is equipped with an external IO-connector. Four optical decoupled inputs and four optical decoupled outputs are available. By default these IO-lines are suited for applications that use a voltage level of 24V. When using the input, your application's line driver must be able to provide at least 8mA. The output line consists of an open collector output which switches to the external supply level when activated. All output lines are pulled with a resistor (R=10kΩ) to the external ground.
Each output transistor is capable of driving a maximum output current of 50mA. Note that you can't get full source voltage $V_{EXT}$ at the output line due to voltage drops caused by the output transistor.

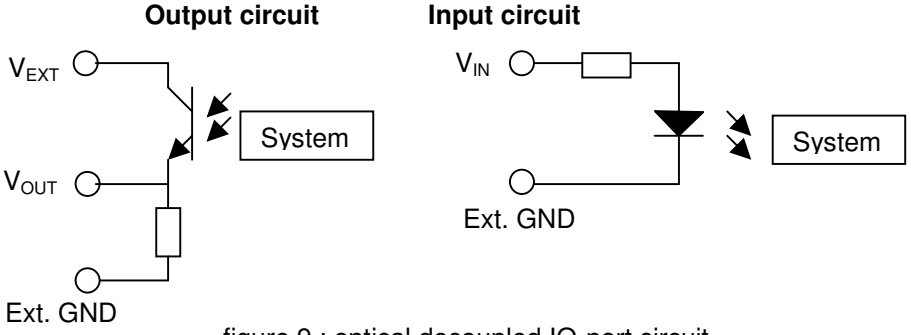



figure 9 : optical decoupled IO-port circuit

Three registers to control the IO-port are mapped into IO-space:

The USER-OUTPUT register at IO-address `0x300` stores the binary pattern which switches the output transistors. A "1" activates the transistor so that the output pin has $V_{OUT} = V_{EXT} - 0.7V$ potential. Writing a "0" disables the transistor and the output pin is pulled to GND.

USER-OUTPUT **IO `0x300` (byte, r/w)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| reserved | | | | OUT3 | OUT2 | OUT1 | OUT0 |
| RESET: | | | | | | | |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

The USER-INPUT register at IO-address `0x301` stores the current state of the input lines.

USER-INPUT **IO `0x301` (byte, ro)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| reserved | | | | IN3 | IN2 | IN1 | IN0 |

To activate the IO-port the IO-enable bit IOEN has to be set. If this bit is set to 0 the whole port ist disabled. That means if the port is disabled the input register stores the last input pattern and writing to the output register stores the bit pattern but the output transistors are not active.

IO-ENABLE **IO `0x302` (byte, r/w)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

| reserved | IOEN |
|---|---|
| RESET: | |
| - | 0 |

IOEN = 0: IO-port disabled
IOEN = 1: IO-port enabled

Register addresses for an optional secondary IO-Port (emPC-X133/2..2U):
`0x310`: USER-OUTPUT
`0x311`: USER-INPUT
`0x312`: IO-ENABLE
All bit functions for the secondary IO-Port are identical compared to the primary IO-Port. Please handle all reserved bit positions as don't care (these bits may be used by future applications or board identification).

For the emPC-X133/2U (2xCAN, 1xUSB) and the emPC-X133/1U the primary IO-Port is replaced by the USB connector. Since the control hardware of the primary IO-Port is still implemented, all IO-Port registers are accessible but have no function!

## 3.3 User programmable Front Panel LEDs

Next to the Power LED there are three LEDs that are programmable by the user. However, some of these LED have special meanings while the system boots. This boot up is controlled by the SrecLoad firmware (see chapter 4: SrecLoad firmware). During the hardware initialisation after powering up the system the red LED is lit. After the initialisation process has finished the red LED goes off and the green LED is lit, indicating that the system is ready to run applications. During a software upload using the SrecLoad the yellow LED flashes to indicate that a data transfer via the serial port is in progress.

The LEDs are connected to the AMD Elan's programmable input / output pins PIO6 (green), PIO7 (yellow) and PIO8 (red). To control the LEDs, two registers are necessary: the PIO-data register (PIODATA15_0) and the PIO direction register (PIODIR15_0). These registers control 16 PIO line but only the three bits 6, 7 and 8 are important to control the LEDs.
Programming hints: for a reliable LED function, set the PIODATA bits 6, 7 an 8 to 0 using a word-wide access but do not modify the other bits !
To turn on a LED set the appropriate bit in the direction register.

Example in x86 assembler:

```
mov    edi, PIODATA15_0          ;load PIODATA15_0 address
mov    ax, [edi]                 ;read content of PIODATA15_0
and    ax, 0fe3fh                ;set mask -> PIODATA6,7,8 = 0
mov    [edi], ax                 ;write back to PIODATA15_0

mov    edi, PIODIR15_0           ;load PIODIR15_0 address
mov    ax, [edi]
or     ax, 0100h                 ;set mask for red LED
mov    [edi], ax                 ;turn on red LED
.
.
.
mov    edi, PIODIR15_0           ;load PIODIR15_0 address
mov    ax, [edi]
and    ax, feffh                 ;set mask for red LED
mov    [edi], ax                 ;turn off red LED
```

USER-LED PIODATA15_0 **0xfffefc30 (word, rw)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | red | yel | grn | reserved | | | | | |

USER-LED PIODIR15_0 **0xfffefc2a (word, rw)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| reserved | red | yel | grn | reserved |
|---|---|---|---|---|

## 3.4 CompactFlash Socket

For data storage the emPC-X133 is equipped with an internal compact flash type I/II connector. As an ordering option a compact flash card is already installed. This interface can be handled like a standard PC-style IDE interface (primary port). The actual ATA registers are implemented in the flash disk itself. The compact flash card must support the “true IDE mode”.
Note: some compact flash card chipsets do not support the IDE LED at the front panel. LED function is not guaranteed.

The ATA command block is mapped in the IO-memory space at address: `0x1f0 .. 0x1f7`
The ATA control block is located at IO-memory address: `0x3f4 .. 0x3f7`

## 3.5 USB Host controller

As an ordering option, the emPC-X133 can be equipped with an USB 1.1 compatible host controller chip (OPTi 82C861). This OHCI compatible chip is directly connected to the internal PCI bus.
Two USB lines are supported by this chip, but only the first line is usable with the emPC-X133. According to the USB 1.1 specification the maximum current consumption over the USB line is limited to 500mA.

It is not recommended to use bus powered USB hubs. Always use a self powered hub, if you want to connect more than one high power USB device ($I_{device}$ > 100mA)

The OPTi 82C861 USB controller chip is supported by Linux. When building the kernel you have to activate the USB support for OHCI compliant devices.

## 3.6 $I^2C$ EEPROM

To store data which must not get lost when the system is powered off, the emPC-X133 has a built in EEPROM. By delivery default, a 24LC04 device with a capacity of 4kBit organized in 2 banks of 256 bytes is installed. The first 16 bytes of bank 0 are reserved by the firmware, so make sure not to overwrite these bytes when programming own applications.
If you use an OS like Linux we suggest to use our **nvpm** software. This is a prep compliant partition manager for non volatile memory devices. Refer to [6] for more information.
Otherwise if you have to program low level software, here are some basic hardware informations about the serial EEPROM. For detailed information about the EEPROM device itself please refer to the 24LC04B datasheet [5].

The EEPROM is directly connected to the Elan’s PIO port pins PIO30 (serial data, SDA) and PIO31 (serial clock, SCL).
PIO31 (SCL) is configured by the firmware as an output (PIODIR31_16 register). To generate a high level signal simply set bit 15 in the PIOSET31_16 register. To set SCL to low level, set bit 15 in the PIOCLR31_16 register. Writing a 0 to each register PIOSET and PIOCLR has no effect on the other PIO pins.
PIO30 (SDA) is the bidirectional serial data line. This line can be driven either by the Elan or the serial EEPROM. Normally this pin is programmed as an input and driven to high level by a pull-up resistor. To generate a low level signal, the data register PIODATA31_16 bit 15 must be set to 0 and bit 15 of the direction register PIODIR31_16 must be set to 1 for output. Switching back to input causes a high level signal.
Before reading this pin for incoming data from the EEPROM set the direction to input (PIODIR31_16 bit 15 → 0). Remember when switching the direction, that the other bits of this register must not be changed !

The incoming data is derived by reading the PIODATA31_16 register bit 14. All other bits should be processed as don't cares.

EEPROM PIOSET31_16 **0xfffefc36 (word, rw)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SCL | reserved, write as 0 | | | | | | | | | | | | | | |

SCL = 0: no effect
SCL = 1: PIO31 (SCL) high output

EEPROM PIOCLR31_16 **0xfffefc3a (word, rw)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SCL | reserved, write as 0 | | | | | | | | | | | | | | |

SCL = 0: no effect
SCL = 1: PIO31 (SCL) low output

EEPROM PIODIR31_16 **0xfffefc2c (word, rw)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SCL_ DIR | SDA_ DIR | reserved, do not modify | | | | | | | | | | | | | |

SCL_DIR = 1: output (do not modify)
SDA_DIR = 0: input
SDA_DIR = 1: output

EEPROM PIODATA31_16 **0xfffefc32 (word, rw)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SDA | | | | | | | | | | | | | | |

SDA_DIR = 1 and SDA = 0: low level output SDA
SDA_DIR = 0: serial data read SDA

**Timing:**
For a reliable function of the serial EEPROM make sure to keep the following timing:

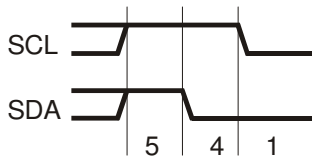
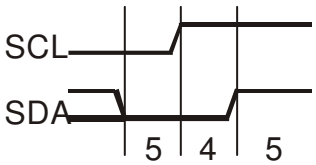
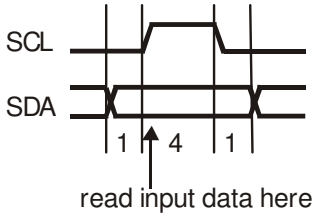
| **Start condition** | **stop condition** | **data transfer** |
|---|---|---|
|  |  |  |
| timing values in µs | | |

table 11: EEPROM timing

Example in C:

```
#define SCL       (1<<15)  /* PIO 31 */
#define SDA       (1<<14)  /* PIO 30 */

int eprom_init(void)
{
    *(volatile unsigned short *)(PIODATA31_16) |= SCL;  /* preset data to 1 */
    *(volatile unsigned short *)(PIODIR31_16)  |= SCL;  /* output */
    *(volatile unsigned short *)(PIODIR31_16)  &= ~SDA; /* input */
    *(volatile unsigned short *)(PIODATA31_16) &= ~SDA; /* 0 */
```

```
    return 0;
}

static void eeprom_start(void)
{
    /* Raise SCL, by programming port bit */
    *(volatile unsigned short *)(PIOSET31_16) = SCL;

    /* Raise SDA, by setting it's port to input */
    *(volatile unsigned short *)(PIODIR31_16) &= ~SDA;
    EEP_DELAY(5);  /* …is declared elsewhere */

    /* Lower SDA, by setting it's port to output */
    *(volatile unsigned short *)(PIODIR31_16) |= SDA;

    EEP_DELAY(4);

    /* Lower SCL, by programming port bit */
    *(volatile unsigned short *)(PIOCLR31_16) = SCL;

    EEP_DELAY(1);
}

static void eeprom_stop(void)
{
    /* Lower SDA, by setting it's port to output */
    *(volatile unsigned short *)(PIODIR31_16) |= SDA;

    EEP_DELAY(5);

    /* Raise SCL, by programming port bit */
    *(volatile unsigned short *)(PIOSET31_16) = SCL;

    EEP_DELAY(4);

    /* Raise SDA, by setting it's port to input */
    *(volatile unsigned short *)(PIODIR31_16) &= ~SDA;
    EEP_DELAY(5);
}

static unsigned char i2c_bit_io(unsigned char sdata    /* output data bit */ )
{
    int            y;

    /* Set output data. */
    if( sdata & 1 ){
         /* port is input -> high state */
         *(volatile unsigned short *)(PIODIR31_16) &= ~SDA;
         EEP_DELAY(5);  /* To allow rise time of SDA */
    }
    else {
         /* port is output -> low state */
         *(volatile unsigned short *)(PIODIR31_16) |= SDA;
    }

    EEP_DELAY(1);

    /* Raise SCL, by programming port bit */
    *(volatile unsigned short *)(PIOSET31_16) = SCL;

    /* Read the data at the SDA line */
    y = (*(volatile unsigned short *)(PIODATA31_16)) & SDA;

    EEP_DELAY(4);

    /* Lower SCL, by programming port bit */
    *(volatile unsigned short *)(PIOCLR31_16) = SCL;

    EEP_DELAY(1);

    return( y ? 1 : 0 );  /* return readout bit */
}
```

| address | Description |
|---|---|
| 0 – 1 | used by autodetect algorithm |
| 2 – 15 | SrecLoad firmware parameters (**don't modify !**) |

| | |
|---|---|
| 16 – (Size-1) | controlled by the partition manager or user definable |

table 12: EEPROM address map

## 3.7 I²C temperature sensor LM75

The emPC-X133 V2.x is equipped with an internal LM75 temperature sensor. This device is connected to the same I²C-bus as the EEPROM (see 3.6). Please refer to [7] for more information about this chip. The LM75's 7 bit slave address is hard wired to 0x49 (binary 1001001) and it's O.S. output (used by the programmable thermal watchdog) is connected to the ELAN's PIO14 line. PIO14 is configured as an input line which can be read via the PIODATA15_0 (0xfffefc30, bit 14) register.
If you have ordered the JANZ LEM Linux installation, a small program called "lm75" can be used to read out and display the actual temperature which is inside the housing of the emPC-X133.

## 3.8 Interrupt mapping and PCI device numbers

| IRQ (ISA) | Description |
|---|---|
| 0 | Timer 1, Counter 0 |
| 1 | Keyboard |
| 2 | Cascade input from Controller 2 |
| 3 | COM2 |
| 4 | COM1 |
| 5 | CAN Controller at 0xd0000 |
| 6 | - |
| 7 | CAN Controller at 0xd1000 (optional) |
| 8 | RTC |
| 9 | CAN Controller at 0xd2000 (optional) |
| 10 | → PCI-IRQ B, D |
| 11 | → PCI-IRQ A |
| 12 | Mouse |
| 13 | FPU error |
| 14 | IDE interface |
| 15 | - |

| IRQ (ISA) | IRQ (PCI) | PCI-Device |
|---|---|---|
| 11 | A | USB controller |
| 10 | B | n.c. |
| 9 | C | n.c. |
| 10 | D | Ethernet Controller |

| PCI Device number | PCI Device |
|---|---|
| 19 | Ethernet Controller |
| 20 | USB controller |

table 13: interrupts and PCI devices

## 3.9 Memory Map

| Address | Description |
|---|---|

| | |
|---|---|
| 0xffff0000 – 0xffffffff | 64 kByte Boot FLASH space (mirrored from 0x3fff0000 – 0x3fffffff) |
| 0xfffef000 – 0xfffeffff | MMCR space, Elan SC520 hardware control register |
| 0x40000000 – 0xfffeefff | dedicated PCI bus space, reserved |
| 0x3e000000 – 0x3fffffff | reserved Boot FLASH space<br>1MB FLASH: 0x3ff00000 – 0x3fffffff<br>2MB FLASH: 0x3fe00000 – 0x3fffffff |
| 0x10000000 – 0x3fdfffff | default PCI bus space |
| 0x00000000 – 0x0fffffff | 256 MB SDRAM space, but only a maximum of 128 MB (0x00000000 – 0x07ffffff) is usable with the emPC-X133 (primary depending on the installed amount of SDRAM, ordering option).<br><br>0xd0000 – 0xd2fff: CAN Controller space |

table 14: system memory map

# 4 SrecLOAD Firmware

This section describes the fail-safe power-on software that is factory programmed to the boot flash. The SrecLOADer is intended to be a „will-ever-work“ solution to downloading programs to the emPC-X133 (either to DRAM or FLASH).

If you already have an operating system running (from FLASH), then there will probably be a high level tool that allows programming of the FLASH more easily or faster (as it loads via Ethernet for example, like **pflash** found in our Linux package). However, if the download did not succeed, then it's time to come back to SrecLOAD.

Besides of the downloading capabilities, the software does some basic board initialisation. The description in this section refers to V2.2 of the SrecLOAD firmware.

## 4.1 Using SrecLOAD

### 4.1.1 Invoking SrecLOAD

Before you enter the SrecLOAD software, make sure to switch your terminal software to 115200 baud. This is the default baudrate, but you may change it within the command mode of the SrecLOAD.
When powering up the emPC-X133, the red LED is on, indicating, that some hardware initialisation is in progress. As soon as the green led is lit the SrecLOAD awaits the transfer of a S-record file. Press [CTRL]-[x] at your console and a simple prompt “>” appears in your terminal window. The SrecLoad firmware has now entered the command mode. Pressing [i] prints some information to the screen:

```
> i
emPC-X133 SrecLOAD V2.2 (Build 415)
CPU    : AMD Elan SC520 step B1
DRAM   : 32MByte
>
```

The SrecLOAD can also be invoked by pressing the reset button at the front panel for more than three seconds. This will force the SrecLOAD to perform no autoboot and the default baudrate is 9600 bps. Note that the contents of the NV-RAM (boot parameters) will not be harmed (see 4.1.3).

```
please release resetbutton...
SrecLOAD drop in!

  #  ####
   #     #
    #     #
   #     #
  #  ####   JANZ Automationssysteme AG

emPC-X133 SrecLOAD V2.2 (Build 415)

Flash AM29LV800T (1024KBytes)
>
```

### 4.1.2 Downloading Programs/Files

Files are downloaded by just sending literal S-record data to the serial port within a autoboot time after power up. If you are in the command mode, type [q] to exit and start listening for a S-record file. While a file is downloaded, SrecLOAD prints a “.” for every 20 received S-record lines. For cases where you do not see terminal output, it will additionally toggle the yellow LED every 20 S-record lines. It will immediately print an error message and turns on the red LED when the checksum of a record fails. Once having detected the termination record, it will start programming the program to FLASH, or jump to the program if you just downloaded to DRAM. When programming of the FLASH was successful, SrecLOAD will light the green LED.

A sample session with SrecLOAD could look as follows:

```
> q
 … exit command mode
….......................................... (896)
Erasing FLASH …b… ok
Programming FLASH …b… ok
```

The download is started as soon as you send a S-record textfile to the serial port, and you see the dots that represent incoming S-records. The total count of all received data records is printed when the termination record is detected, so you can check whether there was a data loss.

The erase and program actions are then indicated and acknowledged on the terminal. The 'b' character indicates that the operation also affects the bootHook structure.

**WINDOWS**
If you have nothing else, then you can still use "Hyper-Terminal". The "Send textfile…" option is exactly what you would want to use.
Since all terminal programs that I used under Windows are slow in downloading text files, you should consider to copy the data to the serial port from a dos-box, as this is considerably faster. A script to do so could look as follows:

```
mode COM2: baud=115200 parity=none data=8 stop=1
rsh sparc -l hugo "cat /vw5.3/target/config/megabox/flash.hex" >d:\tmp\b.txt
type d:\tmp\b.txt >COM2
del d:\tmp\b.txt
```

**UNIX**
You could use any terminal program if you have one installed. If not, you can simply use **cat** to send the data to the serial port. In this case you need to set the terminal line discipline to **–ECHO**, otherwise UNIX will echo the "." input that it reads on the serial port back to the emPC-X133 which will cause S-record failures.

The following sample script does not use the **–ECHO** option, as it was used in parallel to a running terminal program session (on port /dev/ttyS1). That means that the terminal program gets the "." on the screen, as a verification that the download is running. The script does some baudrate switching, as the terminal session uses a different baudrate than SrecLOAD (9600 vs 115200 in this case).

```
#!/bin/sh

TERM=/dev/ttyS1
DEFAULT_PROGRAM=jll.s3
PROGRAM=${1:-$DEFAULT_PROGRAM}

stty 115200 <${TERM}
# Download the file
cat $PROGRAM >${TERM}
# Reboot the system
echo ^X^X >${TERM}
stty 9600 <${TERM}
```

### 4.1.3 Command Mode

A simple command interface was build in the SrecLOAD software to add configuration options and enable some kind of diagnostic. You will enter the command mode by pressing [Ctrl]-[x] when the software is awaiting the download of an S-record.

The following commands are accepted at the command mode prompt (">"):

| Command | Description |
|---|---|
| **"i"** | Print some board and software info. |
| **"c"** | Configure software. This command uses a self explanatory topic driven |

| | |
|---|---|
| | structure. The configuration parameters are stored in the NV-RAM. |
| **"p"** | Print configuration that is maintained by the "c"-command. |
| **"d"** | Display memory. Takes three parameters: address, length and data width. This could be used to check that the FLASH was programmed as you intended it to be. |
| **"@","g"** | Start the FLASH program |
| **"Ctrl-x"** | Restart SrecLOAD by rebooting the emPC-X133 (by hardware reset) |
| **"q"** | Quit, jump to download mode. |
| **"h"** | Prints this command list |

See the following sample for a command mode session:

```
> h
commands:
i  - identify board
c  - configure
p  - print configuration
d  - display memory (addr,count,width)
@  - start program in flash
q  - quit
> i
emPC-X133 SrecLOAD V2.1 (Build 332)
CPU    : AMD Elan SC520
DRAM   : 16MByte
> c
Select Category (use . to abort)
s: Serial port
b: boot options
c: select CPU speed (factory setting, don't modify!)
>> s
Select Baudrate
0: 9600  1: 19200  2: 38400  3: 57600  4: 115200
>>> 4 ok
> p
NV Settings:
Serial Port Baudrate : 115200
Autoboot Timeout: 2
CPU speed: 100 MHz

> d fffef000,10,1
fffef000: 01 00 01 00 ff ff 83 c4 08 8b 45 fc 8b 55 f8 29
>
```

User inputs are emboldened. Single key strokes will invoke the commands (you need not press the return button). Pressing the '.'-key has an abort function that leads back to the command prompt.

#### 4.1.3.1 configuring the boot up process

When in command mode (">") the startup behaviour of the emPC-X133 can be configured. When pressing "c" at the command mode top level, the configuration menu is shown. To select the serial port baudrate, press "s" and select the desired speed. Don't forget to switch your terminal software to the corresponding baudrate, too.

Pressing "b" when in the configure-menu lets you change the boot parameters:

"t": the boot timeout can be adjusted. This is the duration time the SrecLoad waits before starting programs in the flash memory like the ROM-Linux loader. If the time is set to zero, nothing will be auto loaded. The SrecLOAD will enter the command mode using the default baud rate.

"d": a download breaks autoboot (V2.2 or higher): receiving characters via the serial port interrupts the autoboot process. If S-records are detected the download process is activated. This is the default setting corresponding to 4.1.2.

"b": only [CTRL]-X breaks autoboot (V2.2 or higher): the autoboot process can only be interrupted by pressing the [CTRL]-X keys during the defined autoboot timeout. The command mode is entered

and programs can only be downloaded by pressing “q”. Then the SrecLoader waits for incoming S-records.

### 4.1.4 Autoboot

The SrecLOAD is able to autoboot a software in the flash after a predefined time. If there are no S-records received or [CTRL]-[x] is pressed within the autoboot timeout, the program in the flash is started automatically.
If the NV-RAM configuration (use the “p” and “c” commands to check or modify) has a nonzero value for the autoboot-timeout, then SrecLOAD waits for a download on the serial port as it would do normally. If no character is received within the autoboot-timeout (specified in seconds), then SrecLOAD will automatically start the program in FLASH if a valid boot-hook is found. If the autoboot-timeout is set to zero, then no autobooting will be done at all.
For good reasons, SrecLOAD will be absolutely silent on the serial port, when you have configured it for autoboot. It will however give an error message when autoboot is activated by no valid program is found in FLASH. The RUN-LED will be turned to green, when the SrecLOADer is ready to accept a download.

## 4.2 Details about the Download Function

The SrecLOADer is capable of downloading MOTOROLA S-record files through the serial port of the emPC-X133. MOTOROLA S-record files are similar to Intel-Hex files: a file format for transport of binary data in ASCII representation. See chapter 5.3 for more information about MOTOROLA S-record file format.

Basically, there exists three types of records:

| Record Type | Description |
|---|---|
| Start record | Has a descriptive information about the file. The contents are not specified, but most often the name of the object file is stored in this record. Not expected and only used by the SrecLOADer to reset the error status from the last transmission. |
| Data record | A block of data in conjunction with it’s destination address |
| Termination record | Terminates a file (or transaction). Can contain an address to which control should be passed (a start address). |

When SrecLOAD is invoked, it waits for data records, and copies the data to the specified destination address. When it receives a termination record, it *calls* the specified start address. Thus, is possible for the called software to *return* back to SrecLOAD. If you want to restart the software, you only need to retransmit the termination record from the original S-record file (hoping that the program in memory is still intact).

In the basic configuration of the emPC-X133 a FLASH memory with a size of 1 MByte is installed. It contains the SrecLoad firmware located in the upper non erasable 16 kByte block. But as an ordering option, other sizes are planned or already available. Please contact Janz Automation for further information.

In fact the FLASH region starts at memory address `0x3fc00000` and has got a size of 4 MByte (ending at address `0x3fffffff` (1GByte) [1]).
When SrecLOAD detects a destination address that is in FLASH, then it copies all data to a buffer in RAM, and programs all the data to the FLASH once it detects the termination record. Once having programmed the data, it is again ready for the next download.
If the start-address in the termination record is not zero, then this address will be programmed to a bootHook area in the FLASH, so that the downloaded software can be automatically started after a

[1] The upper 64 kByte of the FLASH are also mirrored into the reset segment from `0xffff0000` to `0xffffffff`. Refer to [1] if you need more information.

reset. If the start-address is zero, then the bootHook will not be touched, unless you overwrite it with your own data (from the data records).
Before it can program the data to FLASH, SrecLOAD needs to erase the FLASH. The software is smart enough to only erase those sectors that are covered by the downloaded data (It will erase it, regardless whether this is really required or not: even if all bytes are `0xff`). As it only keeps track with the minimum and maximum address in the S-record file, all gaps in the data file will also be considered as data. Thus, if you want to program some data at the extreme ends of the FLASH, and want to keep the contents in the middle intact, you need to do this in two download sessions.

→Tips: To erase a block in FLASH, it is sufficient to send a short S-record file, that just contains a zero at the first and at the last address of the block to be erased.

When your program runs out of the FLASH it is horribly slow. This might be no problem if your program just copies (or decompresses) itself to DRAM and then jumps there. This is the way the Linux kernel is bootstrapped. If your program does not have this facility, then you can use another trick: SrecLOAD mirrors the BIOS and the BIOS extension segments to DRAM. This effects the last 128KByte of the FLASH. The only thing you need to do, is to modify your S-record in a way that all data records point to the FLASH (at end of 1Gbyte) and leave the termination record to point to the DRAM shadow of the FLASH (at end of 1Mbyte). You can only do this trick for programs that are small enough to fit to the last (128-32)kByte of the FLASH.

**Flash Layout**
In figure 10 we show the sector layout of the FLASH devices that is used on the emPC-X133 and how SrecLOAD makes use of it.

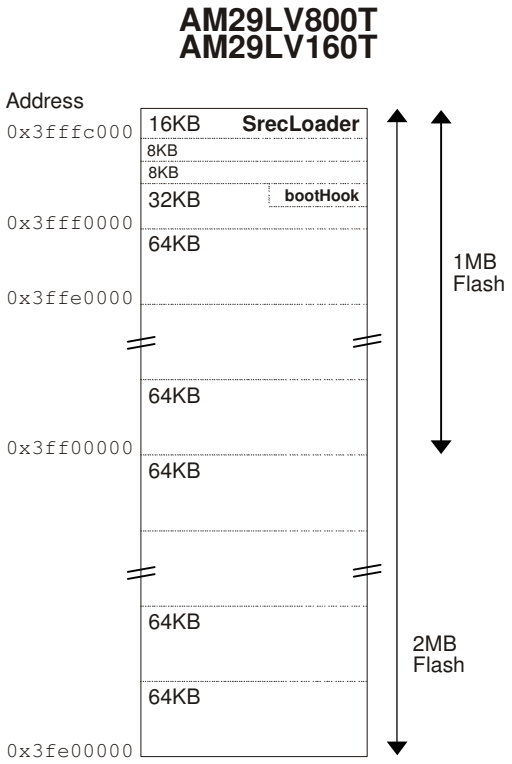



figure 10: FLASH layout

SrecLOAD uses the topmost 16Kbyte sector of the FLASH. Therefore it is not possible to bypass the start of this software after reset. Only this one sector of the FLASH is write-protected. All others are

writeable, and are under full control of the users application, except for the boot-Hook that is described below. This boot hook contains some start-up information for SrecLOAD. It's layout is as follows:

| **Address** | **Name** | **Description** | |
|---|---|---|---|
| 0x3FFF7FFE | MAGIC | A Magic number. BootHook is only valid if this is programmed to 0x3102. This is done automatically when SrecLOAD enters a start address into the START-0 field. | |
| 0x3FFF7FFC | FLAGS | Bits | Description |
| | | 15..2 | Reserved, leave in 1-state for compatibility with further version of SrecLOAD. |
| | | 1 | CBE: Copy-before-execute. If zero, then SrecLOAD copies FLASH data to DRAM before execution. |
| | | 0 | Reserved, leave in 1-state for compatibility with further version of SrecLOAD. |
| | | The flags field is only erased by SrecLOAD when your S-record has a nonzero termination record. It is never written to. If you want some flags to be set (to zero), then this information must be part of your S-record data. | |
| 0x3FFF7FF8 | START-0 | SrecLoad will call this address (after it's board initialisation) when it detects the correct pattern in the MAGIC field. Can be any address, might be targeted to DRAM if CBE mode is used. If address is in FLASH range (1GByte-*flash_size*…1Gbyte-1), then the correct FLASH bank is selected before the jump is done. | |
| 0x3FFF7FF4 | CPY_LEN | Number of bytes to copy when CBE is used. | |
| 0x3FFF7FF0 | CPY_SRC | Flash source address for CBE mode. Must be in range 1GByte-*flash_size*…1GByte-1. | |
| 0x3FFF7FEC | CPY_DEST | DRAM destination address for CBE mode. Must be in range 0..*dram_size-1*. | |
| 0x3FFF7FEB | reserved | reserved, don't touch by your S-record | |
| : | : | : | |
| 0x3FFF7FE0 | reserved | reserved, don't touch by your S-record | |

table 15: SrecLoad boot hook details

# 5 Appendices

## 5.1 Technical Data

**CPU**

| | |
|---|---|
| Type | AMD Elan SC520 (with integrated peripherals) |
| Clock Frequency | 133 MHz |
| Bus Clock Frequency: | Memory bus: 66 MHz, PCI bus 33 MHz |
| L1-Cache | 16 kB unified cache (write-through or write-back) |
| integrated FPU | IEEE 754 compliant |

**Memory**

| | |
|---|---|
| SDRAM | up to 128Mbyte (32bit memory bus) |
| Boot FLASH | 1Mbyte, 3,3V in-circuit programmable (up to 2MByte) |
| NVRAM | 114Bytes (located in RTC) with battery backup<br>4 kBit (512 bytes) located in $I^2C$ EEPROM, 16 bytes reserved by the system |
| Storage | internal 50pin Compact flash type I/II socket for CompactFlash cards (ATA compatible “true IDE mode”) |

**Interfaces**

| | |
|---|---|
| Ethernet | AMD AM79C973 10/100Mbit PCnet-PCI II Ethernet controller (bus master capable) with 12Kbyte SRAM FIFO. 10/100BaseT port at front panel. |
| Serial Interface | 16550 compatible UART (COM1), on front panel D-SUB connector. Baud rate up to 115200 bps. RS232 level.<br>COM1 signals: RxD, TxD, RTS, CTS, DCD, DTR, DSR, RI |
| CAN | on front panel D-SUB male connector (according to CiA DS-102)<br>Controller: SJA 1000<br>ISO/DIS 11898, opto-isolated (500V)<br>software switchable bus line termination (120 Ohms) |
| User I/O | 4x opto-decoupled input lines (max. 24V input)<br>4x opto-decoupled output lines (max. 50mA switching capability per line)<br>Isolation : 500V |
| USB | front panel type A receptable<br>USB 1.1 PCI host controller OPTi 82C861 |

**Expansion Capabilities (optional)**

| | |
|---|---|
| internal expansion slot | additional ports (CAN, serial, I/O, …)<br>(“extended” version) |

**Power Requirements**

| | | |
|---|---|---|
| Input Voltage | 10 .. 28 V AC / DC | |
| emPC-X133 version | **basic** | **/2, /3 version** |
| Power (max.) | 6 W | 8W |

**Physical Dimensions**

| | | |
|---|---|---|
| Height | 76 | 76 |
| Width | 52 | 65 |
| Depth | 120 | 120 |
| Weight | 397 g | 496 g |

**Environmental Specifications**

| | |
|---|---|
| Temperature range | 0..+50°C (operating), -20..+75 (non operating) |
| Humidity | 0%..80%, non condensing |

| | |
|---|---|
| ESD protection | 2KV (human body model) |

## 5.2 References

These references direct you to manuals and specifications, that you might need to know when you attempt to program the emPC-X133. Most of the documents can be downloaded from the Internet. Look for the WWW servers of the chip manufacturers.

[1] **Élan SC520 Microcontroller User's Manual #22004A**, AMD Inc., 1999
[2] **PCI local bus specification**, PCI Special Interest Group, Revision 2.1, 1995.
[3] Datasheet: **SJA 1000 Stand-alone CAN controller,** Philips Semiconductors, 1997.
[4] **Élan SC520 Microcontroller Register Set Manual #22005A**, AMD Inc., 1999
[5] Datasheet: **24LC04B/08B, #DS21051H**, Microchip Technology, 2001
[6] **ROM Linux Loader**, Janz Automatiossysteme AG, 2001
[7] Datasheet: **LM75**, National Semiconductor, 1999
[8] Datasheet: **82C861 Fire Link**, OPTi Inc, Rev. 1.0, 1998
[9] CF+ and CompactFlash spec. Rev 1.4, CompactFlash Association 1999

The following titles are books for more in depth reading. If a book is in the list, it does not explicitly say that it is a good one. However, there is no book that we haven't learned something from …

[7] Brey, Barry B.: **The Intel 32-bit microprocessor: 80386, 80486, and Pentium**, Prentice Hall, 1995.
[8] Messner, Hans-Peter: **PC Hardware Buch – Aufbau, Funktionsweise und Programmierung**, Addison Wesly, 5. Auflage, 1997

**WWW-References**

| | |
|---|---|
| Janz Computer AG: | www.janz.de |
| Advanced Micro Devices: | www.amd.com |
| Microchip Technology | www.microchip.com |
| Philips Semiconductor | www-us.semiconductors.philips.com |

# 5.3 MOTOROLA S-Record Format

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

### 5.3.1 S-Record Content

When viewed by the user, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data and checksum. Each byte of binary data is encoded as a 2-cacharacter hexadecimal number; the first character representing the high-order 4bits, and the second the low-order 4 bits of the byte.

The five fields which comprise an S-record are shown below:

| Type | Record Length | Address | Code/Data | Checksum |
|---|---|---|---|---|

Where the fields are composed as follows:

| Field | Printable Characters | Contents |
|---|---|---|
| Type | 2 | S-records type. S0, S1, etc. |
| Record-length | 2 | The count of the character pairs in the record, excluding type and record length. |
| Address | 4,6 or 8 | The 2-, 3- or 4-byte address at which the data field is to be loaded into memory |
| Code/Data | 0 – 2n | From 0 to n bytes of executable code, memory-loadable data, or descriptive information For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record). |
| Checksum | 2 | The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address and the code/data fields. |

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems. An S-record file is a normal ASCII text file in the operating system in which it resides.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

### 5.3.2 S-Record Types

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation and decoding functions. The various upload, download and other records transportation control programs, as well as cross assembler, linkers and other file-creating or debugging programs, utilise only those S-records which serve the purpose of the program. For specific information on which S-records are supported by a particular program, the user's manual for the program must be consulted. SrecLOAD support S1, S2, S3, S7, S8, S9 records and gracefully ignores S0 and S5 records.

An S-record format module may contain S-records of the following types:

S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. The address field is normally zero (SrecLOAD assumes an address length of 2bytes)

S1 A record containing code/data and the 2-byte address at which the code/data is to reside.

S2 A record containing code/data and the 3-byte address at which the code/data is to reside.

S3 A record containing code/data and the 4-byte address at which the code/data is to reside.
S5 A record containing the number of S1, S2 and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.
S7 A termination record for a block of S3 records. The address field may optionally contain the 4-byte address of the instruction to which control is passed. There is no code/data field.
S8 A termination record for a block of S3 records. The address field may optionally contain the 3-byte address of the instruction to which control is passed. There is no code/data field.
S9 A termination record for a block of S3 records. The address field may optionally contain the 2-byte address of the instruction to which control is passed. There is no code/data field.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3 or 4 byte address. Normally only one header record is used, although it is possible for multiple header records to occur.

### 5.3.3 S-Records Creation

S-records format files may be produced by dump utilities, debuggers, linkage editors, cross assemblers or cross linkers. Several programs are available for downloading a file in S-record format from a host system to a microprocessor-based system.

### 5.3.4 Example

Shown below is a typical S-record format module (though a bit short), as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The module consists fo one S0 record, four S1 records, and one S9 record.

The S0 record is comprised of the following character pairs:

| | |
|---|---|
| S0 | S-record type S0. Indicating that it is a header record |
| 06 | Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow |
| 00<br>00 | Four character, 2byte, address fields; zero in this example |
| 48<br>44<br>52 | ASCII H,D and R -> „HDR“ |
| 1B | The checksum |

The first S1 record is explained as follows:

| | |
|---|---|
| S1 | S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address. |
| 13 | Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow |
| 00<br>00 | Four-character, 2-byte, address field; hexadecimal address 0000, where the data which follows is to be loaded. |
| 28<br>:<br>:<br>:<br>:<br>:<br>:<br>:<br>:<br>7C | The next 16 character pairs of the first S1 record are the ASCII bytes of the actual program code/data. In this MC68000 assembly language example, the hexadecimal opcodes od the program are written in sequence in the code/data fields of the S1 records:<br>285F move.l (a7)+,a4<br>245F move.l (a7)+,a2<br>2212 move.l (a2),d1<br>226A0004 move.l 4(a2),a1<br>24290008 move.l function(a1),d2<br>237C move.l #forcefunc,function(a1) |
| 2A | The checksum of the first S1 record |

The second and third S1 records also each contain 0x13 (19) character pairs and are ended with checksums 0x13 and 0x52 respectively. The fourth S1 record contains 7 character pairs and has a checksum of 0x92

The S9 record is explained as follows:

| | |
|---|---|
| S9 | S-record type S9. Indicating that it is a termination record |
| 03 | Hexadecimal 06 (decimal 6), indicating that three character pairs (3 bytes) follow |
| 00<br>00 | The address field, zeros. Our program starts at zero. |
| FC | The checksum of this termination record |

Each printable character in an S-record is encoded in a hexadecimal (ASCII in this example) representation of binary bits which are actually transmitted. For example, the start of the first S1 record above is sent as:

| Type = “S1” | | Length = 0x13 | |
|---|---|---|---|
| ‘S’ | ‘1’ | ‘1’ | ‘3’ |
| 0x53 | 0x31 | 0x31 | 0x33 |

## 5.4 Product History

Note that changes in the major version number are related to a PCB redesign. Though, PCB redesign need not be related to functional changes, but might have been done for manufacturing purposes only.

| Version | Release Date | Name | Changes |
|---|---|---|---|
| V0.9 | 02.04.2001 | hr | • released for internal software development |
| V1.0 | 24.09.2001 | | • initial version |
| V2.0 | 10.10.2002 | hr | • new CPU board design with CF and USB → V2.0 |
| V2.0 | 02.02.2003 | hr | • new expansion board with additional CAN/IO ➔ emPC-X133/2 |
| V2.1 | 15.10.2004 | hr | • Bootflash AM29LV160 is now default |

## 5.5 Manual History

| Version | Release Date | Name | Changes |
|---|---|---|---|
| V0.1 | 18.09.2001 | hr | draft, for internal development. |
| V1.0 | ??.??.???? | | initial version |
| V1.1 | 18.02.2002 | hr | SrecLoader V2.2 options added |
| V2.0 | 07.11.2002 | hr | now refers to hardware version 2.0 |
| V2.1 | 14.02.2003 | hr | includes information about emPC-X133/2 (additional CAN/IO) |
| V2.2 | 23.01.2004 | hr | new ordering options and front panel views, USB infos added |
| V2.3 | 14.10.2004 | hr | Bootflash AM29LV160 is now default |
| V3.0 | 06.05.2004 | hr | replaced name “CAN-megaBox” with “emPC-X133” |